ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH011

BUILDING CONTROL SYSTEM REMOTELY

M. Lukaszewski*, K. Klys, E9 Controls Limited, London, England

Abstract

Building a control system for a scientific facility is a complex process that requires significant time coordination and the cooperation of hundreds of people. The latest of our control system for the 10 J 100 Hz laser system happened to be built when access to the lab was far more difficult, due to the pandemic: travel was much more complex, and local restrictions at one point prevented a large number of people from being in the lab simultaneously. This paper shows how, despite the demanding working conditions, we built the control system for the 10 J 100 Hz laser. The control system was designed in collaboration with CLF (Central Laser Facility, Science and Technology Facilities Council, UK) and HiLASE (Institute of Physics, Czech Academy of Sciences). The process of building the laser control system was divided into stages and we had to rely on remote working. This article discusses how we adapted each stage to work remotely, what tools we used, how we minimized risks, and what we would have done differently if we had started from scratch.

INTRODUCTION

This paper has been written to share the methods and tools used to build a control system for the 100 Hz 10 J laser system built in collaboration with CLF (Central Laser Facility, Science and Technology Facilities Council, UK) and HiLASE (Institute of Physics, Czech Academy of Sciences). The paper aims to share findings and processes, and evaluate risk awareness.

REMOTE WORK

The main engineering effort started shortly after the first wave of pandemic restrictions, thus making it impossible to work directly in the laboratory. Experience with previous deployments was leveraged, and it was decided that the core integration layer be built remotely without accessing the actual devices. This approach was possible because proper software practices were followed from the beginning, such as: expressive documentation, simulation of various system components, continuous integration during development, automation of repeatable tasks and open communication with stakeholders. All of these aspects will be discussed below.

RISK MITIGATION

Building a scientific system is an arduous and lengthy process. It can be broken down into individual stages, including design, tendering, subsystem construction and equipment integration. The process ends with the implementation of the control system and functional testing of the entire system.

* marcin.lukaszewski@e9controls.com

However, problems with tenders, broken supply chains and other factors affecting the work process, which are not the subject of this article, cause numerous delays that increase the risk of error during integration and negatively affect the developer experience.

To mitigate the risks arising from delays, several strategies have been implemented to react effectively in a rapidly changing environment. It should be noted that a large proportion of the problems that arise during the development of software for controlling equipment are due to a lack of adequate information or late acquisition of such information.

Involvement Therefore, it is essential to participate in the design work, where the parameters of each device are detailed. At this point, each change is "cheap and easy" to make, which provides an opportunity for finding errors regarding device communication and with possible impact on the quality of the integration of devices into the control system.

Regular Contact Also, direct and regular contact with equipment suppliers allows changes in the control software to be tracked, potentially affecting the integration efforts.

Device Grouping Off-the-shelf equipment that was widely available, or was from large suppliers was given the lowest priority. It was assumed that basic integration for these devices would be available in the environment or that the system would be relatively simple to build, given the availability of documentation.

Devices used in the past, or with similar functionality to those already deployed were given medium priority. It was assumed highly likely that software for such devices either already existed in the community, or could be used after modification.

The highest priority was given to new equipment, which was previously unknown and also from suppliers unfamiliar to the laboratory. Equipment from small suppliers was also included in this group. These devices were integrated first, which resulted in the broadest possible time window to develop a working version.

INFRASTRUCTURE

Production Environment – Laboratory

The control system infrastructure in the laboratory consists of five powerful servers that control data collection and devices. The servers are connected via a high-speed Ethernet network to which all devices in the system are connected. The servers run under Linux Ubuntu LTS. The software controlling the system is based on the Experimental Physics and Industrial Control System (EPICS) framework, described later in this article. The lab has other systems with a similar

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH011

framework, so it was decided to build a system with similar functionality to reduce maintenance efforts.

Staging Environment

By definition, the staging environment should replicate the production environment - the laboratory set-up. A staging site's main purpose is to ensure that all new changes are working as intended before they are deployed on the production servers [1]. With this approach we can limit to a minimum the risk of any software failure in the laboratory environment. At this stage we can detect any memory leaks, excessive CPU or RAM usage or just wrong functioning of the software.

The biggest challenge was to reflect the devices that the laser control system is composed of. To do it, we decided to write custom simulators. Their role was to emulate the connection protocols and behaviour of the specific devices. The emulators allow for IOC functioning verification. Furthermore, the staging environment contains the exact copy of the production monitoring and archiving infrastructure.

Testing Environment

The main task of the test servers is to build and perform test executions of the newly created software components. In terms of the control system software for the 100 Hz laser facility, it was crucial to test all EPICS components - IOCs. In the beginning, two test servers were established using one of the cloud providers. They contained all the necessary software to build EPICS together with support modules and to run/ and test IOCs. The custom tool launches IOCs and sets and reads values of PVs.

To be sure that each IOC has been evaluated successfully, we integrated them with Gitlab Continuous Integration (CI) pipelines. Once a new commit appears in the IOC repository, the GitLab shared runner copies the code, builds it, and runs tests on one of the servers.

Beacuse most of the time, the servers have been waiting for new commits and desire to limit costs, we decided to use Docker images containing test tools and the EPICS environment in GitLab and run new commit changes inside the containers. The main disadvantage of this approach is the total test time. GitLab needs time to download images and all dependencies for each test, so it takes more time to perform all of the actions. Nevertheless, it is not the most important feature for users. It is still acceptable if a test takes from 15 to 30 seconds. Thanks to that approach we do not need to maintain remote servers and we can apply the Infrastructure as Code (IaC) idea, which is the whole configuration in Dockerfile.

EPICS

EPICS is a software framework that provides a set of toolkits destined for designing distributed control systems. It is widely used in large experiments like particle accelerators and telescopes, as well as smaller infrastructures such as lasers or industry systems. EPICS employs Client/Server and Publish/Subscribe mechanisms to handle communications between the various computers. The typical EPICS architecture of the control system is organized in a way that a group of servers - IOCs perform real-time actions (e.g. data acquisitions) and then, using dedicated network protocols, Channel Access (CA), or its newer and faster implementation PV Access (PVA), the data are published to subscribed clients [2, 3].

Compared with other control system packages, EPICS does not model control system devices as objects but rather as data entities that describe a single aspect of the process or device under control, thus the name Process Variable (PV) [2].

EPICS is open-source software and one of its main advantages is the community that constantly improves it and creates helper tools that facilitate control system development and EPICS integration.

DEVICE SIMULATORS

Device simulators are a key element of the testing and staging environments. They allow for IOC tests in the cases of communication protocols and functional tests. This is a crucial aspect in terms of remote IOC development without physical equipment. Based on the documentation we can test the communication and how the IOC should behave in advance.

We chose to use Lewis. This is a Python package that makes it easy to develop complex stateful device simulations [4]. It supports basic industrial communication protocols like TCP and Modbus and it can establish an EPICS server to expose PVs with device parameters. The package does not support custom communication protocols or low-level drivers, but the 100 Hz laser control system is based on devices with standard protocols TCP (e.g. laser rack, delay generator), Modbus (e.g. PLC), or EPICS interface (e.g. Front-End ModBox) [4].

It is worth mentioning that it is not always possible to simulate all the device states'. In most cases, it is more than enough if we can emulate and test the communication between an IOC and a simulator. After that, when there is a possibility to examine an IOC with a real device, it is much easier and faster to implement the IOC parts corresponding to the device states and behavior while having the proper communication ready. That is why there usually is no need to browse and study the documentation to investigate how the device behaves.

In the described 100 Hz laser infrastructure, the Lewis simulators are used to test IOCs in GitLab CI pipelines. Inside the Docker image, the custom tool launches the IOCs with the corresponding Lewis simulators and performs test cases described in the YAML configuration file. In the staging environment, the simulators are launched together with the IOCs to reflect the laboratory setup.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH011

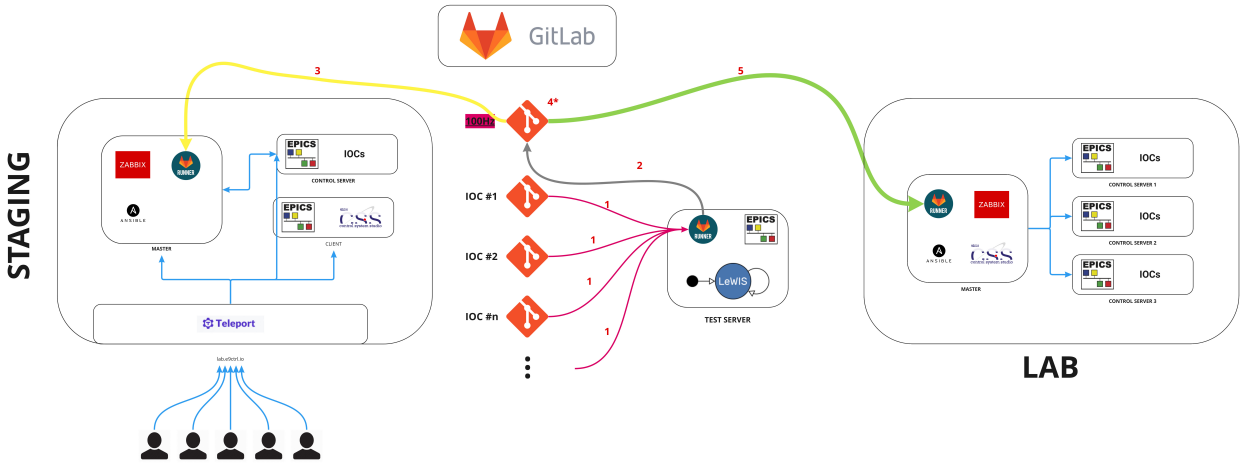

Figure 1: Diagram showing the relationships between the various elements of the system.

AUTOMATION AND CI/CD

Automation is an important aspect of modern IT (information technology) infrastructure. It is the process of creating software and systems to perform repeatable actions and replace or reduce manual intervention. Its main advantage is the acceleration of IT infrastructure delivery by automating processes that previously required a human touch [5].

To limit the risk of mistakes and ease long-term maintenance, we have automated almost all processes: infrastructure build, testing, deployment, and monitoring. This has drastically minimized human engagement, increased system responsiveness, and improved user experience.

Continuous integration is a coding philosophy and set of practices that drives development teams to frequently implement small code changes and check them into a version control repository [6]. We have applied CI practices to IOC testing. Each commit is verified in terms of syntax rules, formatting, and functional tests being performed.

Continuous delivery picks up where continuous integration ends and automates application delivery to selected environments. It is an automated way to push code changes to any environment [7]. We have designed the workflow that tests IOCs and when a new commit appears in the master branch, deploys them to the staging environment. It is presented in the Fig. 1.

Building Infrastructure

To avoid manual server configuration and to facilitate server management, we decided to employ Ansible. For each environment (staging and production) there is a specific playbook with the corresponding list of hosts and variables to set up the proper configuration, i.e. network settings, installation of all required software, and starting of needed processes. The Ansible playbooks are also the infrastructure documentation, and thanks to them, it is easier to follow any configuration modifications.

For tests, the Docker image has been designed. It contains the EPICS framework and all the tools to perform tests inside the container when the GitLab CI pipeline is launched.

Tests

As mentioned in the previous section, the tests are made automatically, after each new commit in the repository. They are performed using GitLab CI pipelines together with custom Docker images, and they are divided into several jobs. The first one checks if all needed files are available - files describing test cases, IOC, and simulator directories. Then, the IOC database files are validated in terms of syntax errors. After that, the IOC is built and installed. The last step is the functional tests. Using the YAML configuration file with the test cases (modify set point value and verify its readback, or check if the PV value has changed over a defined time), the custom called 'ioc-test' performs functional tests by launching IOC together with the simulator and setting and reading PVs characterized in the YAML file.

Deployment

If the IOC tests have been finished successfully, the GitLab CI triggers the deployment job. Its role is to launch the Ansible playbook with the IOC deployment procedure based on the source of the trigger (IOC type). The Ansible role checks if there is a new IOC version in its repository. If yes, it installs it on the staging server together with a new simulator instance. The IOC is automatically added to the monitoring and achiever engine. This is a user who decides when to start the new IOC. We want to avoid running the same IOC in two various versions and users receiving alarms from the killed, old IOC version during deployment.

MONITORING

System and application monitoring is a vital IT function that has a wide range of benefits for various businesses and facilities. It can save money on network performance, em-

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH011

ployee productivity, and infrastructure costs, and it is far more strategic than its name implies [8].

The monitoring can be divided into basic types depending on their purpose:

1. Availability monitoring, which concerns server management and services status;
2. Web and network performance monitoring, which are tools that can detect excessive network load, a high number of requests to a single service, or any network bottleneck; and
3. Application monitoring, which tracks the performance of an application and spots any issues before they cause service failure [9].

An effective monitoring system has many features from which developers and infrastructure users can profit. Monitoring services can not only prevent incidents but also allow you to detect them faster when they do happen. Fast incident detection results in time and money savings. Monitoring servers and systems improve the use of the hardware [8]. We can take advantage of all hardware resources if we know that they are in the operational state. We can predict network traffic or excessive CPU usage by some of the applications or services and try to optimize hardware architecture. In terms of early issue detection, it is important to provide a real-time notification [8]. They will not only alert you about performance issues but also make it easy for you to resolve those issues. It can be done via emails, text messages, or any other integration with communication applications (e.g. Slack). Finally, effective monitoring enhances the end-user experience. If infrastructure is slow, it will result in many support calls from end users. Delivering top-of-the-line IT-enabled services can improve the productivity of the systems and increase the number of satisfied end users [8, 10].

Laser Monitoring System

The designed monitoring system is composed of many elements for both servers and IOC monitoring. We have created the custom Go client that uses CA protocol (using the specially designed Go implementation of EPICS CA library) that monitors PVs from the IOC. We focused on PVs from the IocStats module like IOC CPU, RAM usage and Heartbeat, which informs about IOC state, but also PVs that inform about the connection state between IOC and the device (asynRecord) and some IOC-specific PVs like the number of acquired frames from camera IOC. The Go client exposes metrics about the number of monitoring PVs and their state (connected/disconnected).

The client sends data to VictoriaMetrics. It is an open-source data series database typically used for processing high volumes of data and for long-term data storage [11]. It uses 10x less RAM than InfluxDB and up to 7x less RAM than Prometheus, Thanos, or Cortex when dealing with millions of unique time series. It can be used as a drop-in replacement for Prometheus because it supports Prometheus querying API [11].

For visualisation, we chose yet another open-source tool - Grafana. It helps to create queries and visualize application performance metrics with customized data. The tool lets one create monitoring dashboards for metrics over a specific period, so it can easily be adapted for a specific project [12].

To monitor the hardware, we decided to use the Prometheus node exporter, which exposes metrics with all essential server parameters. Those are scraped directly to VictoriaMetrics and visualized on the available dashboard templates prepared by the community.

The alerting system is based on the Prometheus Alertmanager and vmalert. We prepared a set of alert rules in PromQl about hardware metrics and IOC metrics - for example, an alert should be raised when the IOC heartbeat stops growing or when the PV with the device connection state changes its value. To provide real-time messages we redirected alarms to one of the Slack channels. In the error message, one can see the level of the alert (error, warning, resolved), the metric name and its value, the host origin, and a short metric description.

DOCUMENTATION

Documentation is an essential part of building control systems, regardless of whether the build is based on remote working or a traditional build. In designing the documentation, we have tried to follow current trends to ensure is easily accessible, is up-to-date, covers as much context as possible, and is easily maintainable and modifiable in the future.

Key elements of the system have been recorded in the form of Architecture Decision Record (ADR) documents, which contain important architectural decisions together with context and consequences. Each document has a number and relates to the one problem it describes. Examples of such documents include those describing the network in the system or the EPICS PV naming scheme.

Another important documentation element is a network map with addresses and information on the location of individual servers and computers used to operate the system. All logins and passwords are stored in a password manager.

A document containing information on each device used in the system is also essential. This includes information on the device, the manual, and contact with the people responsible for the software on the supplier end.

Information about what software is running on which server is contained in the deployment configuration files, which are an integral part of the main repository with the system code. Hence, additional documentation of these is not necessary.

CONCLUSIONS

We have successfully delivered the fully operational infrastructure for the 100 Hz laser control system. It follows modern management practices (CI/CD, IaC, multiple environments) and limits human intervention to a minimum by general automation of the processes. Automated IOC testing, together with unmanned deployment to the staging environment, makes the control system less prone to errors and allows for fast integration with existing applications.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH011

Thanks to the monitoring that covers not only server parameters but also IOC statistics, we can effectively react to failures and try to prevent them. We have proved that the research infrastructure can be similar to state-of-the-art bank applications or web applications in terms of testing, deployment, and monitoring.

There are still areas that we are working on and trying to improve. We are currently developing our implementation of the device simulators. Since the internal state of the device is not a crucial part of the simulators from a testing point of view, we decided to limit that aspect and focus on easier protocol implementation that would allow a user to describe any non-standard communication type. This will make IOC testing less time-consuming and more effective. What is more, together with laser scientists, we are developing more user-friendly monitoring dashboards, benefiting from their knowledge and experience.

ACKNOWLEDGEMENTS

The 100 Hz laser system was funded by European Regional Development Fund and the state budget of the Czech Republic project HiLASE CoE (CZ.02.1.01/0.0/0.0/15-006/0000674); Horizon 2020 Framework Programme (H2020) (739573).

REFERENCES

[1] *Why should you use Staging Environment?*, https://apiumacademy.com/blog/why-should-you-use-staging-environment/ (accessed on 2022-09-20)

[2] M. Kraimer, J. Anderson, A. Johnson, W. Norum, J. Hill, R. Lange, B. Franksen, P. Denison, and M. Davidsaver, "EPICS Application Developer's Guide", EPICS Base Release 3.15.5., https://epics.anl.gov/base/R3-15/5-docs/AppDevGuide.pdf (accessed on 2022-09-20)

[3] EPICS about, https://epics-controls.org/about-epics/ (accessed on 2022-09-20)

[4] Lewis documentation, https://lewis.readthedocs.io/en/latest/index.html (accessed on 2022-09-20)

[5] A. Axelrod, "Complete Guide to Test Automation", 1st ed. edition, Apress, 2018, pp. 3-4.

[6] M. Labourady, "Pipeline as Code: Continuous Delivery with Jenkins, Kubernetes, and Terrafom", Manning, 2021, pp. 3-5.

[7] M. R. Pratama and D. Sulistiyo Kusumo, "Implementation of Continuous Integration and Continuous Delivery (CI/CD) on Automatic Performance Testing", 2021 9th International Conference on Information and Communication Technology (ICoICT), 2021, pp. 230-235.

[8] S. Lagus, "Effective Monitoring and Alerting: For Web Operation", 1st edition, O'Reilly Media, 2012, pp. 1–5.

[9] *What Is IT Monitoring?*, https://www.splunk.com/en_us/data-insider/what-is-it-monitoring.html (accessed on 2022-09-20)

[10] J. Hernantes, G. Gallardo, and N. Serrano, "IT Infrastructure-Monitoring Tools", *IEEE Software*, vol. 32, no. 4, pp. 88-93, July-Aug. 2015. doi:10.1109/MS.2015.96

[11] VictoriaMetrics documentation, https://docs.victoriametrics.com/ (accessed on 2022-09-20)

[12] Grafana documentation, https://grafana.com/docs/ (accessed on 2022-09-20)